

# 案例 12-区域八邻接点边界填充算法

文档编写：霍波魏

校稿/修订：孔令德

时间 2019~2020

联系方式：QQ997796978

**说明：**本套案例由孔令德开发，原版本为 Visual C++6.0，配套于孔令德的著作《计算机图形学-基于 MFC 三维图形开发》一书。孔令德计算机工程研究所的学生霍波魏在学习计算机图形学期间，对本套案例进行了升级并编写了学习文档。现在程序的编写和程序的解释都是基于 Windows 10 操作系统，使用 Microsoft visual studio 2017 平台的 MFC（英文版）开发。

## 一、知识点

八邻接点：对于多边形区域内部的任意一个种子像素，其左、上、右、下、左上、左下、右上、右下这八个像素称为八邻接点。

## 二、案例描述

本案例用区域八邻接点泛填充算法，填充“孔”字。

## 三、实现步骤

1. 添加栈结点 CStackNode 类。
2. 在 CTestView 类中定义出/入栈函数、泛填充函数以及双缓冲绘图函数。
3. 在 CTestView 类的 BoundaryFill8 函数中使得上、下、左、右、左上、左下、右上、右下八个像素入栈。
4. 在 CTestView 的构造函数中对新旧颜色进行初始化，旧颜色即原来的颜色，新颜色即待填充颜色。
5. 在 DrawGraph 函数中绘图并在 OnDraw 函数中进行调用。

## 四、主要算法

```
CTestView 类
public:
    void DrawGraph(CDC* pDC); //绘制图形
    void BoundaryFill8(); //泛填充算法
    void Push(CPoint point); //入栈
    void Pop(CPoint &point); //出栈
protected:
```

```

int nClientWidth, nClientHeight; //屏幕客户区宽度和高度
int nHWidth, nHHeight; //屏幕客户区的半宽和半高
COLORREF OldClr, NewClr; //旧颜色为区域的原色, 新颜色为填充色
CStackNode* pHead, *pTop; //结点指针
CPoint Seed, Left, Top, Right, Bottom, LeftTop,
        RightTop, RightBottom, LeftBottom; //种子及其八个邻接点
void CTestView::DrawGraph(CDC* pDC) //绘制图形
{
    CRect rect; //定义客户区
    GetClientRect(&rect); //获得客户区大小
    nClientWidth = rect.Width(); //屏幕客户区宽度
    nClientHeight = rect.Height(); //屏幕客户区高度
    nHWidth = nClientWidth / 2; //屏幕客户区半宽
    nHHeight = nClientHeight / 2; //屏幕客户区半高
    CDC memDC;
    memDC.CreateCompatibleDC(pDC);
    CBitmap NewBitmap, *pOldBitmap;
    NewBitmap.LoadBitmap(IDB_BITMAP1);
    pOldBitmap = memDC.SelectObject(&NewBitmap);
    BITMAP bmp;
    NewBitmap.GetBitmap(&bmp);
    int nX = rect.left + (nClientWidth - bmp.bmWidth) / 2;
        //计算位图在客户区的中心点
    int nY = rect.top + (nClientHeight - bmp.bmHeight) / 2;
    pDC->BitBlt(nX, nY, nClientWidth, nClientHeight, &memDC, 0, 0, SRCCOPY);
    memDC.SelectObject(pOldBitmap);
    NewBitmap.DeleteObject();
}
void CTestView::BoundaryFill8() //八邻接点填充
{
    CDC* pDC = GetDC();
    pHead = new CStackNode; //建立栈头结点
    pHead->pNext = NULL; //栈头结点的指针域为空
    Push(Seed); //种子像素入栈
    if (BkClr != pDC->GetPixel(Seed.x, Seed.y) && BoundaryClr
        != pDC->GetPixel(Seed.x, Seed.y))
    {
        while (pHead->pNext != NULL) //如果栈不为空
        {
            CPoint PopPoint;
            Pop(PopPoint);
            if (SeedClr == pDC->GetPixel(PopPoint.x, PopPoint.y))
                continue;
            pDC->SetPixelV(PopPoint.x, PopPoint.y, SeedClr);
        }
    }
}

```

```

    Left.x = PopPoint.x - 1; //搜索出栈结点的左方像素
    Left.y = PopPoint.y;
    COLORREF CurPixClr; //当前像素的颜色
    CurPixClr = pDC->GetPixel(Left.x, Left.y);
    if (BoundaryClr != CurPixClr && SeedClr != CurPixClr)
        //像素不是边界色并且未置成填充色
        Push(Left); //左方像素入栈
    LeftTop.x = PopPoint.x - 1; //搜索出栈结点的左上方像素
    LeftTop.y = PopPoint.y + 1;
    CurPixClr = pDC->GetPixel(LeftTop.x, LeftTop.y);
    if (BoundaryClr != CurPixClr && SeedClr != CurPixClr)
        Push(LeftTop); //左上方像素入栈
    Top.x = PopPoint.x; //搜索出栈结点的上方像素
    Top.y = PopPoint.y + 1;
    CurPixClr = pDC->GetPixel(Top.x, Top.y);
    if (BoundaryClr != CurPixClr && SeedClr != CurPixClr)
        Push(Top); //上方像素入栈
    RightTop.x = PopPoint.x + 1; //搜索出栈结点的右上方像素
    RightTop.y = PopPoint.y + 1;
    CurPixClr = pDC->GetPixel(RightTop.x, RightTop.y);
    if (BoundaryClr != CurPixClr && SeedClr != CurPixClr)
        Push(RightTop); //右上方像素入栈
    Right.x = PopPoint.x + 1; //搜索出栈结点的右方像素
    Right.y = PopPoint.y;
    CurPixClr = pDC->GetPixel(Right.x, Right.y);
    if (BoundaryClr != CurPixClr && SeedClr != CurPixClr)
        Push(Right); //右方像素入栈
    RightBottom.x = PopPoint.x + 1; //搜索出栈结点的右下方像素
    RightBottom.y = PopPoint.y - 1;
    CurPixClr = pDC->GetPixel(RightBottom.x, RightBottom.y);
    if (BoundaryClr != CurPixClr && SeedClr != CurPixClr)
        Push(RightBottom); //右下方像素入栈
    Bottom.x = PopPoint.x; //搜索出栈结点的下方像素
    Bottom.y = PopPoint.y - 1;
    CurPixClr = pDC->GetPixel(Bottom.x, Bottom.y);
    if (BoundaryClr != CurPixClr && SeedClr != CurPixClr)
        Push(Bottom); //下方像素入栈
    LeftBottom.x = PopPoint.x - 1; //搜索出栈结点的左下方像素
    LeftBottom.y = PopPoint.y - 1;
    CurPixClr = pDC->GetPixel(LeftBottom.x, LeftBottom.y);
    if (BoundaryClr != CurPixClr && SeedClr != CurPixClr)
        Push(LeftBottom); //左下方像素入栈
}
}

```

```

else
    MessageBox(_T("请在空心字体内部单击鼠标左键!"), _T("提示"));
delete pHead;
pHead = NULL;
ReleaseDC(pDC);
}
void CTestView::Push(CPoint point)//入栈函数
{
    pTop = new CStackNode;
    pTop->PixelPoint = point;
    pTop->pNext = pHead->pNext;
    pHead->pNext = pTop;
}
void CTestView::Pop(CPoint &point)//出栈函数
{
    if (pHead->pNext != NULL)
    {
        pTop = pHead->pNext;
        pHead->pNext = pTop->pNext;
        point = pTop->PixelPoint;
        delete pTop;
    }
}

```

## 五、实现效果

区域八邻接点边界填充算法效果如图 12-1 所示。



图 12-1 区域八邻接点边界填充算法效果图

## 六、案例心得

该填充算法和区域四邻接点边界填充算法类似，只需要把搜索方式由四邻接点修改为八邻接点即可，四邻接点可以填充的区域八邻接点都可以填充，但反之

则不一定；如遇到狭窄区域，四邻接点无法填充，而八邻接点则可以通过左上、左下、右上、右下四个区域去填充，进而穿过狭窄区域填充整个图形。